SWITCH e INTRODUCCIÓN A LOOPS

Laboratorio de Programación

If else UN DATO MAS

Existe un atajo se llama ***operador ternario***. Es más aplicable cuando una declaración if / else devuelve un valor, pero también puede funcionar de otra manera. Un operador ternario se ve así:

```
if (a < b) {
  min = a;
} else {
  min = b;
}
```

```
min = a < b ? a : b;
```

else if

La sentencia else if siempre viene después de la sentencia if y antes de la sentencia else (si la hay). Al igual que la instrucción else, las declaraciones else if también son siempre opcionales. El else if entonces toma una condición, y puedes tener más de una de ellas. He aquí un ejemplo.

```c
int main() {
    int anosExperiencia;
    if (anosExperiencia == 9) {
        printf("Principiante\n");
    } else if (anosExperiencia == 0) {
        printf("Junior\n");
    } else if (anosExperiencia == 2) {
        printf("Semi senior\n");
    } else if (anosExperiencia == 5) {
        printf("Senior\n");
    } else {
        printf("Super Senior\n");
    }
}
```

Complejo? Entonces usemos un switch!

switch

Los programas con múltiples resultados son tan comunes que C proporciona una declaración especial para ellos: Switch. Una declaración de ***Switch*** proporciona una sintaxis alternativa que es más fácil de leer y escribir. Sin embargo, los encontrará con menos frecuencia que if y else.

```c
int main() {
    int grado = 1;
switch (grado) {
  case 1:
    printf("Bipedo\n");
    break;
  case 2:
    printf("Novato\n");
    break;
  case 3:
    printf("Junior\n");
    break;
  case 4:
    printf("Senior\n");
    break;
  default:
    printf("Invalid\n");
    break;
}
}
```

En el ejemplo anterior, el valor o la expresión de la sentencia switch es ***grado***. Una restricción de esta expresión es que debe evaluarse como un tipo integral *(int, char, short, long, long long o enum)*. Dentro del bloque, {}, hay varios casos. La palabra clave case verifica si la expresión coincide con el valor especificado que viene después.

La palabra clave ***break*** le dice a la computadora que salga del bloque y no ejecute más código ni verifique ningún otro caso dentro del bloque de código. Si ninguno de los casos es verdadero, se ejecutará el código de la instrucción ***default***.

Cuidado!!: Sin la palabra clave ***break*** al final de cada caso, el programa ejecutaría el código para el primer caso coincidente y todos los casos posteriores, incluido el código predeterminado. Este comportamiento es diferente de las declaraciones condicionales if / else que ejecutan solo un bloque de código.

LOOPs o BUCLES

Un bucle es una forma de repetir código hasta que se alcanza una condición específica. Por ejemplo, ¿alguna vez has estado buscando a alguien en una tienda de comestibles? Revisas cada pasillo hasta que los encuentras. Naturalmente, realiza ciertas tareas con una repetición o iteración, hasta que se cumple una condición. En este caso, ¡cuando finalmente logras vislumbrar a una persona familiar!

Ese término genérico "***iterar***" significa "***repetir***". ¡Descubrirá que, al igual que en la vida real, usará bucles en su código todo el tiempo! Y al igual que este GIF infinito de bloques girando, una tarea puede repetirse aparentemente infinitamente, pero siempre tienes una forma de controlar cuándo se detiene.

Cuando vemos que un proceso tiene que repetirse varias veces seguidas, escribimos un bucle. Los bucles nos permiten crear código eficiente que automatiza procesos para crear programas escalables y manejables.

En esta lección, aprenderemos sobre tres tipos de bucles: bucles ***while***, bucles ***do-while*** y bucles ***for***.

while

El bucle ***while*** se parece mucho a una sentencia if. Y al igual que una declaración if, ejecuta el código dentro de ella si la condición es verdadera. La diferencia, sin embargo, es que el ciclo while continuará ejecutando el código dentro de él, una y otra vez, siempre que la condición sea verdadera. Aquí hay una forma simple del ciclo while:

```
while (condicion) {
  // código
}
```

En otras palabras se ejecuta mientras se cumpla la condición

Ejercicio 1

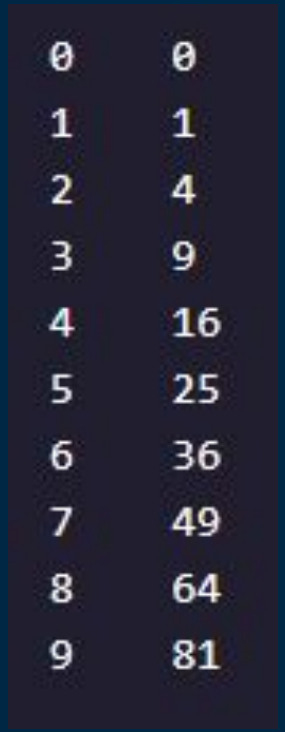
Realice un programa que conceda acceso al sistema cuando se ingrese el nro correcto.

```c
int main() {

  int respuesta = 0;
  printf("Ingrese nro de acceso ");
  while (respuesta != 8) {
  printf("Respuesta erronea, intenta de nuevo: ");
  scanf("%d", &respuesta);
}
 printf("ACCESO CONSEGUIDO");

    return 0;
}
```

Ejercicio 2



El primer programa que se ejecutó en una computadora como programa almacenado (el EDSAC). Fue escrito y ejecutado por David Wheeler en el laboratorio de computación de la Universidad de Cambridge, Inglaterra, el 6 de mayo de 1948, para calcular e imprimir una lista simple de exponentes como la imágen.

Puede realizar un loop que logre dicho output?

do while

Ahora que hemos escrito un ciclo while, escribamos una forma alternativa. Primero veremos su sintaxis y luego explicaremos qué está sucediendo y por qué:

```
do{
        // Declaraciones)
} while (condición);
```

Esto es extraño, la condición vino después de las declaraciones. ¿Significa esto que la condición no se verifica hasta que se ejecutan las declaraciones? ¡Eso es exactamente lo que sucede!

Esto es lo que se conoce como bucle do-while. En pocas palabras, primero hace algo y luego verifica la condición y repite de esta manera hasta que la condición ya no es cierta.

El ciclo do-while se usa con mayor frecuencia cuando un programa quiere hacer algo al menos una vez antes de verificar la condición.

Ejercicio 3

Modifique el siguiente programa para que imprima los números decrementales entre 10 y 0:

```c
int main() {

    int i = 0;

    do {
        printf("%d\n", i);
        i++;
    } while (i < 11);
}
```

for loop

Los ejemplos de bucle while que hemos visto hasta ahora iteran sobre una secuencia de números. Esto es tan común que C, como la mayoría de los otros lenguajes de programación, tiene una sintaxis especial para ello. Cuando sabemos exactamente cuántas veces queremos iterar (o contar), podemos usar un bucle ***for*** en lugar de un bucle while. Por ejemplo, este es un bucle for:

```
for (int i = 0; i < 10; i++) {
  printf("%d\n", i);
}
```